Universidade de São Paulo - ICMC
Bacharelado em Ciências de Computação
SSC0103 - Programação Orientada a Objetos
Prof. Márcio Delamaro
Aluno PAE: Misael Junior

Especificação do Software Xadrez

Gabriel Simmel Nascimento Nº USP: 9050232
Marcos Cesar Ribeiro de Camargo Nº USP: 9278045
Victor Luiz Roquete Forbes Nº USP: 9293394

São Carlos - SP
23 de Junho de 2016

1 Introdução

O projeto consiste na implementação do jogo de Xadrez online com interface gráfica. Foram implementadas todas as regras de Xadrez incluindo todos os empates possíveis e jogadas como Roque, En Passant e Promoção. O jogo foi implementado baseado na notação *Forsyth-Edwards Notation* (FEN).

O programa foi implementado usando Java 8. Explicaremos nesta documentação as decisões tomadas e como as funções foram implementadas. O programa foi compilado e testado no sistema operacional Linux (Ubuntu 16.04 LTS e Manjaro 16.06).

2 Objetivos

O objetivo principal do trabalho foi desenvolver o jogo completo de Xadrez para uma partida entre duas pessoas. Além do objetivo principal, foi proposto desenvolver as seguintes funcionalidades:

- Interface Gráfica.
- Inteligência Artificial.
- Partida em LAN contra outro jogador.
- Voltar um movimento em uma partida contra a IA.
- Salvar o estado atual de jogo em uma partida.

3 Descrição do Software

3.1 Classes principais

Para o funcionamento do jogo, necessitamos de 4 classes principais:

- Client: Classe de comunicação entre o usuário e o servidor. Classes auxiliares permitem que a comunicação seja feita via interface gráfica.
- ClientAI: Uma classe que herda da classe Client, mas que tem acesso à mecânica do jogo, para que possa escolher suas jogadas automaticamente.
- Server: Classe que interpreta os comandos enviados pelos Clientes, valida-os e os executa, a partir da classe ChessBoard.
- ChessBoard: Classe principal da mecânica do jogo. Guarda uma matriz 8x8 de peças, que são, por sua vez, objetos de uma classe abstrata chamada ChessPiece.

Há uma classe abstrata diferente para cada tipo de peça (Peão, Cavalo, Bispo, Torre, Rainha e Rei). As classes que identificam uma peça (não abstrata) são aquelas que possuem o nome do seu time seguido do tipo da peça. Essas possuem métodos de, por exemplo, retornar qual é a sua cor e sua representação textual.

Para o time branco, por exemplo, possuímos as classes WhitePawn, WhiteKnight, WhiteBishop, WhiteRook, WhiteQueen e WhiteKing, que herdam de Pawn, Knight, Bishop, Rook, Queen e King, respectivamente. Essas últimas, com a exceção da classe Pawn, implementam o movimento de cada peça, pois os movimentos de um Cavalo, Bispo, Torre, Rainha e Rei são iguais para ambos os times. Como o movimento do peão branco é um pouco diferente do movimento do peão preto, seus movimentos estão implementados nas classes WhitePawn e BlackPawn respectivamente.

3.2 Modos de jogo

Todos os jogos (sendo eles entre jogadores ou contra a IA) são executados em um servidor. Portanto, para iniciar um jogo é preciso que o servidor esteja aberto. O Cliente não possui acesso à mecânica do jogo, isto é, o tabuleiro (ChessBoard.class).

Com o servidor aberto, basta que um usuário se conecte a ele através de um cliente. Ao se conectar, o usuário pode selecionar uma de 4 opções.

- Player vs Player: O primeiro jogador se conecta ao servidor pelo IP e aguarda o segundo jogador se conectar.
- Player vs IA: O jogador se conecta ao servidor pelo IP e um novo Cliente IA é instanciado, se conectando ao servidor logo em seguida.
- IA vs IA: As duas IAs são instanciadas e se conectam ao servidor.
- Carregar jogo: Inicia-se um jogo do tipo Player vs IA a partir de uma partida que ainda não foi finalizada a partir do menu da Figura 1. As partidas são salvas automaticamente.

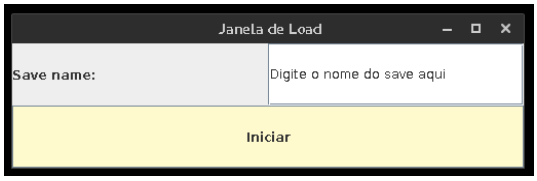

Figura 1: Janela de Load.

3.3 Funcionamento do jogo

Após ambos os clientes conectarem ao servidor, o canal de comunicação do usuário com o jogo passa a ser a interface gráfica. Basta o usuário clicar em uma das casas do tabuleiro (que são JButtons) que o código da casa será enviada para o servidor.

O servidor possui um objeto da classe ChessBoard para poder realizar os comandos enviados pelos usuários. Para instanciar um objeto dessa classe, é necessário fornecer uma string na notação *FEN*. Caso o usuário queira carregar um jogo salvo, o cliente envia essa string para o servidor para poder instanciar um tabuleiro diferente do tabuleiro inicial.

Ao construir um objeto da classe ChessBoard, todos os movimentos possíveis são gerados e armazenados nas respectivas peças. Cada peça possui um função diferente de geração desses movimentos.

Ao chamar a função MakeMove() da classe ChessBoard, o movimento (passado por parâmetro) é executado, todas as informações do tabuleiro são atualizadas e novos movimentos são gerados.

Durante a partida, o estado do jogo é salvo em um arquivo pela classe Client, para que o usuário possa retomar uma partida contra uma IA que foi interrompida.

3.3.1 Modo de uso

Para iniciar o servidor, é necessário executar o arquivo "GUIServer.java" do pacote chess.GUI, abrindo então uma janela com as opções de fechar e iniciar o servidor como ilustra a Figura 2. O cliente pode ser aberto pelo arquivo "Main.java" do mesmo pacote, apresentando então o menu com as interfaces do jogo como ilustra a Figura 3.

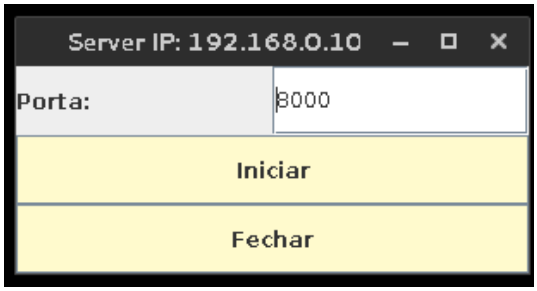

Figura 2: Menu do Servidor.

Figura 3: Menu do Jogador.

Quando é a vez do usuário de jogar, ele terá que primeiro selecionar (clicar) em uma de suas peças. Ao receber o código da casa clicada, o servidor verifica se é uma casa que possui uma peça do usuário. Se possuir, o servidor retorna para o cliente uma lista dos possíveis movimentos daquela peça para que suas casas de destino possíveis sejam pintadas de azul como ilustra a Figura 5.

Caso o servidor receba uma casa de destino inválida, o cliente retorna para o primeiro estágio (de seleção da peça que ele deseja mover).

3.4 Interface Gráfica

Para melhor interação entre o usuário e o programa, foi implementada uma interface gráfica utilizando java.swing. Para facilitar o tratamento de exceções, o usuário está limitado ao uso dessa interface, por meio dos 64 botões dispostos na janela como ilustra a Figura 4.

Figura 4: Estado inicial do tabuleiro.

Figura 5: Casas pintadas de azul.

3.5 Cliente e Servidor

O Servidor, quando iniciado, dispara uma *Thread* que agurda um cliente se conectar. Quando o cliente se conecta, sua conexão é colocada em uma fila, até que o tamanho da fila seja maior ou igual a 2, pois para acontecer um jogo são necessários dois jogadores. A partir do momento que dois clientes se conectam no servidor, um objeto do tipo *Game* é instanciado e iniciado.

Um Cliente, ao se conectar no servidor, aguarda até que haja outro jogador disponível para jogar. Quando isso acontece é retomada a comunicação dos clientes com o servidor, enviando os comandos conforme informado pelos usuários em cada cliente. A cada rodada o Cliente salva em um arquivo o estado atual do tabuleiro, usando a notação *FEN*. O arquivo é salvo com o seguinte formato de nome: Nome do Jogador-day-dd/mm/yy-hh:mm:ss.save.

Um *Game* é uma partida entre dois Clientes. A classe *Game* é uma *Thread* que contém o loop princípal do jogo, incluindo todos os mecanismos necessários para o jogo de Xadrez. Ela requisita dos clientes as informações necessárias para o jogo acontecer, como por exemplo um movimento a ser realizado. A cada rodada o *FEN* do tabuleiro para os clientes.

A Inteligência Artificial também é implementada na forma de um Cliente, na qual as jogadas são escolhidas pelo computador.

4 Imagens

Figura 6: Um jogo de IA vs IA em andamento.

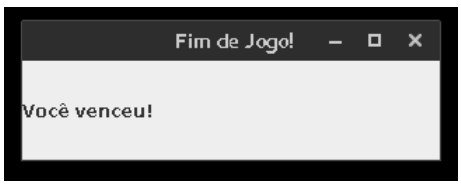

Figura 7: Final de uma partida.

5 Bibliotecas utilizadas

- java.util.TreeSet - Utilizado para armazenar os movimentos de cada peça.
- java.util.HashMap - Utilizado para armazenar o tabuleiro em notação FEN para checar a regra do empate por tripla repetição.
- java.util.Random - Utilizado para deixar a IA imprevisível.
- java.net.Socket - Utilizado para estabelecer uma conexão do cliente com o servidor.
- java.util.Queue - Utilizado para manter em fila os clientes conectados no servidor.
- java.util.LinkedList - Utilizado como fila.
- java.net.ServerSocket - Utilizado para estabelecer as conexões com os clientes.
- java.util.Calendar - Utilizado para salvar o arquivo de jogadas com a data e hora do ínicio do jogo.
- java.util.Date - Utilizado para gerar uma string com a data formatada.
- java.util.Scanner - Utilizado como canal de entrada nos clientes e servidores.
- java.io.PrintStream - Utilizado como canal de saída nos clientes e servidores.
- java.io.BufferedWriter - Utilizado para realizar a leitura do arquivo com as jogadas.
- java.io.FileWriter - Utilizado para escrever o arquivo com as jogadas.

6 Contribuição por aluno

Os autores por código estão definidos no topo de cada arquivo .java, mas decidimos por quantificar a contribuição de cada aluno da seguinte maneira:

- Gabriel Simmel - 30%
- Marcos Camargo - 35%
- Victor Forbes - 35%

7 Referências

- Java API